

Aula – 4

Variáveis e Entrada de Dados

Disciplina: CCO016 - Fundamentos de Programação

Prof: Phyllipe Lima
phyllipe@unifei.edu.br

Universidade Federal de Itajubá – UNIFEI
IMC – Instituto de Matemática e Computação

O que já vimos?



- ☐ Processo de Compilação/Execução
- ☐ Ambientes de Programação para C
 - ☐ VSCode, WSL, Replit
- ☐ Olá Mundo – Meu Primeiro Programa
 - ☐ printf()

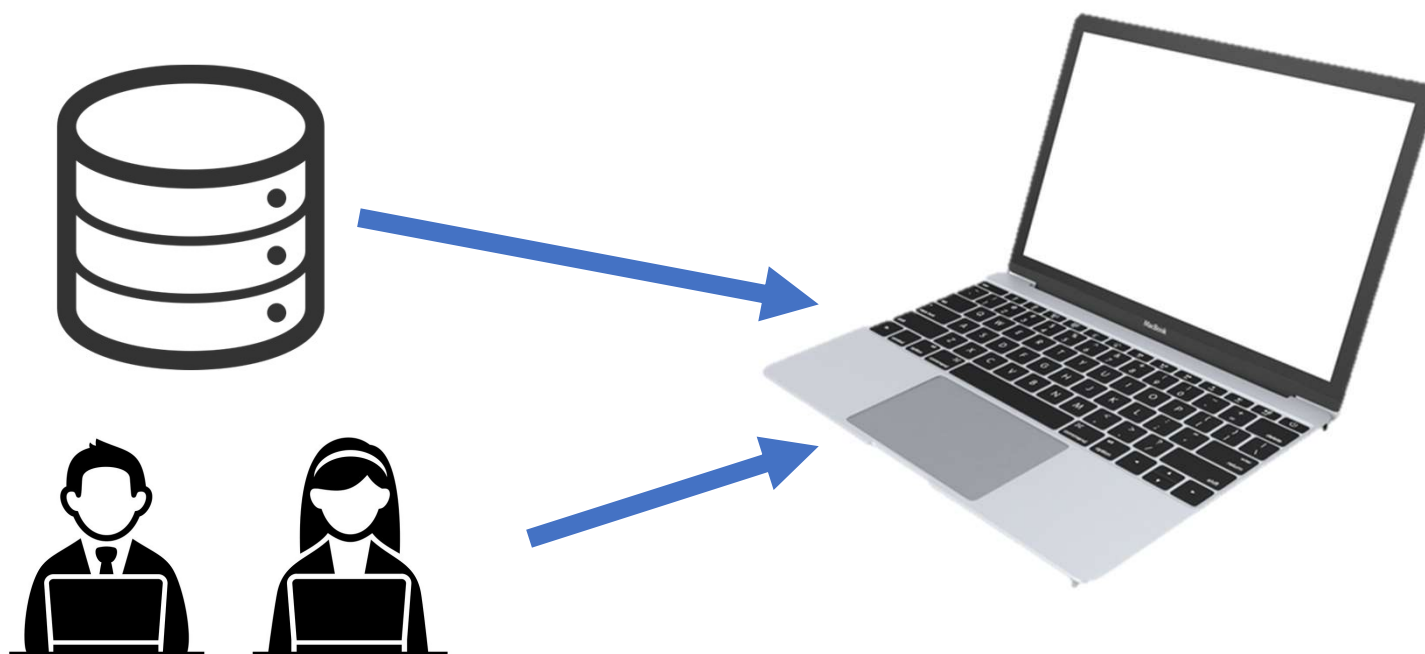
Agenda



- ☐ Variáveis
- ☐ Tipos
- ☐ Entrada de Dados
- ☐ Operadores aritméticos
- ☐ Funções Matemáticas



Como passar dados para
um computador?





Variáveis

Variáveis

- ❑ Precisamos, primeiro, ter algum lugar para guardar as informações.
- ❑ Para isso utilizamos variáveis.

Variáveis - Baldes

- ❑ Podemos pensar em variáveis como “baldes” na vida real. Onde guardamos dados de nosso interesse.
- ❑ Podemos ter baldes de diversos tamanhos e cores.
- ❑ Mas considere que os baldes possuem “tipos”



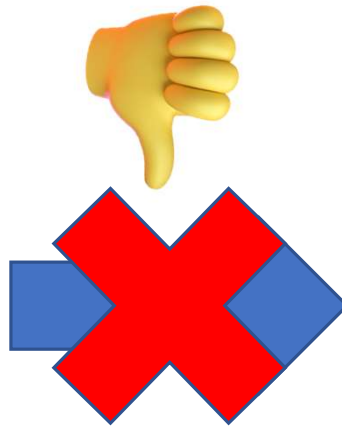
Variáveis - Baldes

- ❑ Se, **ao comprar o balde**, ele é do tipo “pano de chão” ele só pode armazenar “pano de chão”.



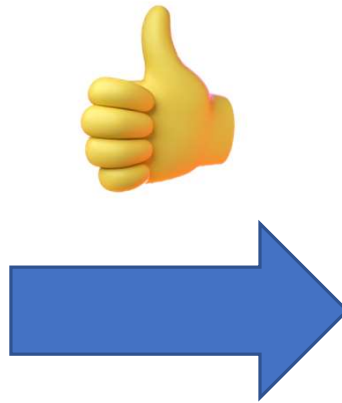
Variáveis - Baldes

- ❑ Se, **ao comprar o balde**, ele é do tipo “pano de chão” ele só pode armazenar “pano de chão”.
- ❑ Não poderá guardar “tênis velho” nesse balde.



Variáveis - Baldes

- ❑ É necessário construir/comprar outro balde do tipo “tênis velho”.
- ❑ Agora sim!



Variáveis - Baldes



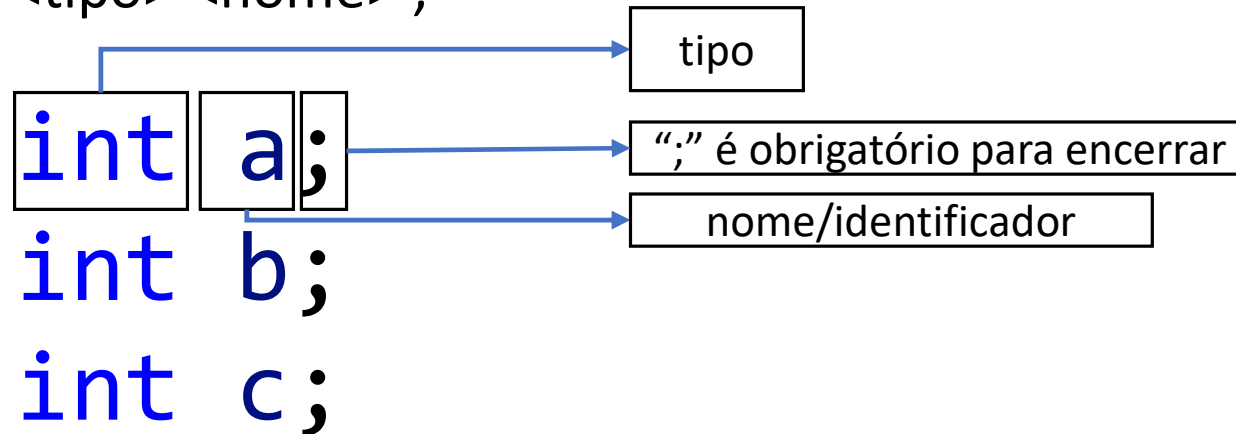
Variáveis – Baldes - Tipos

- ❑ Dizemos que os baldes são “tipados”. Isto é, só podem armazenar dados do tipo que foram declarados (no caso dos baldes, *fabricados*).
- ❑ Em algumas linguagens de programação, como a Linguagem C, o funcionamento é semelhantes.
- ❑ Dizemos que estas são linguagens *fortemente tipadas*.

Declarando variáveis em C

- ❑ Seguimos a seguinte sintaxe:

- ❑ `<tipo> <nome> ;`



- ❑ No exemplo declaramos três variáveis, chamadas `a`, `b` e `c`.
- ❑ Elas são do tipo `int` e só podem armazenar números inteiros.

Declarando variáveis em C

- ❑ Podemos declarar e atribuir valor no mesmo comando. Para isso utilizamos o operador de atribuição “=” (1 igual)
 - ❑ <tipo> <nome> = <valor>;

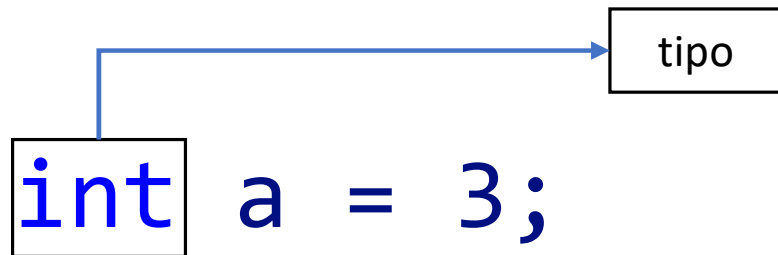
```
int a = 3;
```

- ❑ Para facilitar você pode ler o operador “atribuição” como “recebe o valor de”.
- ❑ a “recebe o valor de” 3

Declarando variáveis em C

- ❑ Podemos declarar e atribuir valor no mesmo comando. Para isso utilizamos o operador de atribuição “=” (1 igual)

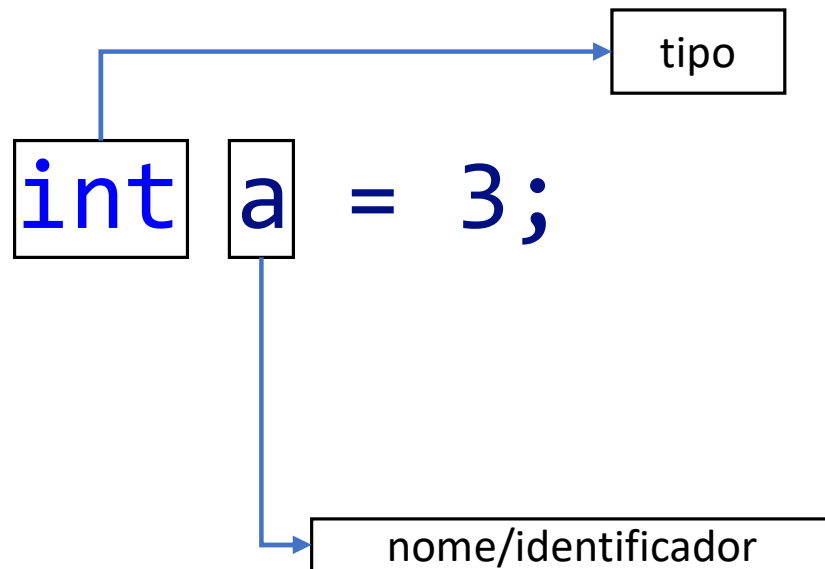
- ❑ <tipo> <nome> = <valor>;



Declarando variáveis em C

- ❑ Podemos declarar e atribuir valor no mesmo comando. Para isso utilizamos o operador de atribuição “=” (1 igual)

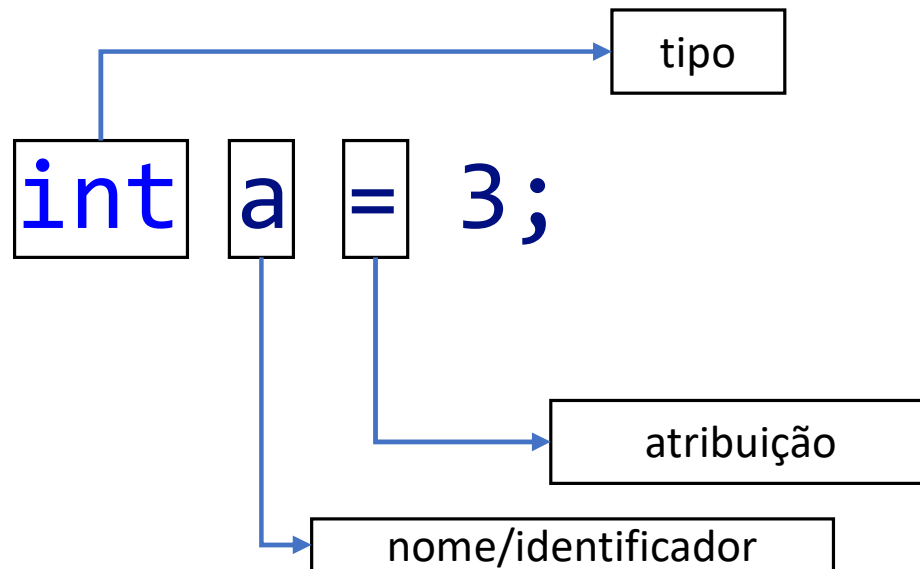
❑ <tipo> <nome> = <valor>;



Declarando variáveis em C

- ❑ Podemos declarar e atribuir valor no mesmo comando. Para isso utilizamos o operador de atribuição “=” (1 igual)

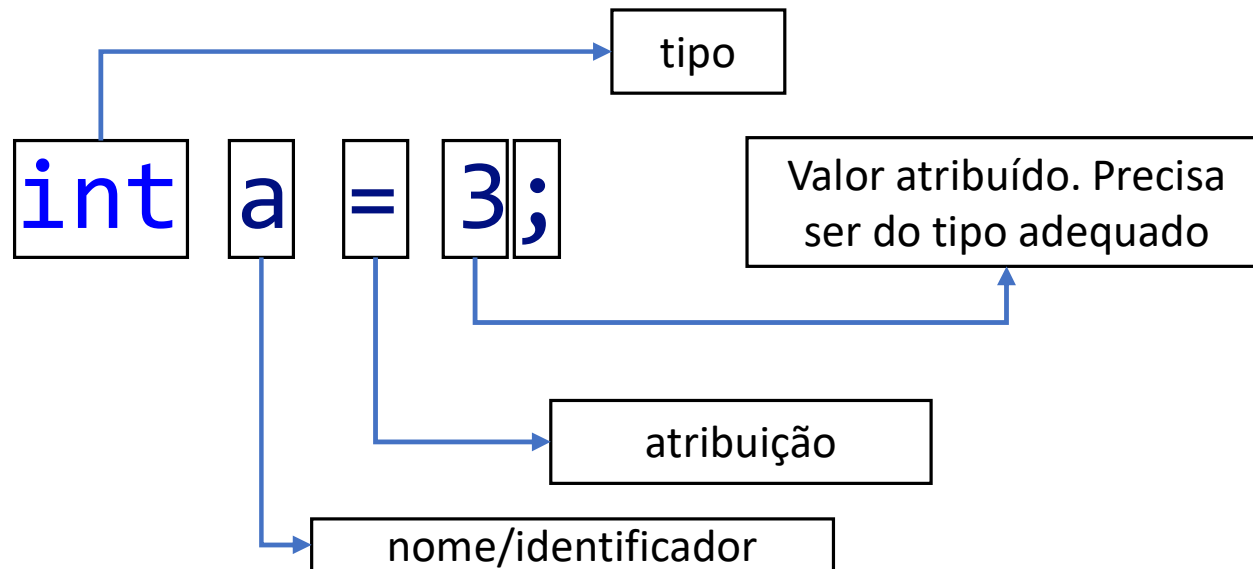
❑ <tipo> <nome> = <valor>;



Declarando variáveis em C

- Podemos declarar e atribuir valor no mesmo comando. Para isso utilizamos o operador de atribuição “=” (1 igual)

`<tipo> <nome> = <valor>;`



Atribuições

- ❑ Podemos declarar e atribuir em comandos diferentes. Mas para cada comando é necessário um “;” (ponto e vírgula)

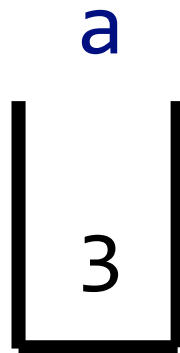
```
int a; //Declaração da variável 'a'  
a = 3; //Atribuir 3 na variável 'a'
```

- ❑ Perceba que na linha de baixo não colocamos “int” novamente. Isso caracterizaria uma nova declaração. Desejamos utilizar a variável chamada “a” e não declarar outra.

Atribuições

- ❑ Após realizarmos uma atribuição é como se tivéssemos “tacado” um objeto dentro do balde.
- ❑ Do ponto de vista da computação dizemos que uma área de memória foi reservada. Essa área pode ser identificada como “a” e lá armazenamos o número inteiro 3.

```
int a;  
a = 3;
```



Área de memória identificada por “a”

Modificações

- ❑ Podemos substituir esse valor quantas vezes desejarmos. Basta realizarmos novas atribuições. Ao longo de um programa uma variável possui seu valor modificado diversas vezes.

```
int a;
```

```
a = 3;
```



Modificações

- ❑ Podemos substituir esse valor quantas vezes desejarmos. Basta realizarmos novas atribuições. Ao longo de um programa uma variável possui seu valor modificado diversas vezes.

```
int a;
```

```
a = 3;
```

```
a = 6;
```



O conteúdo armazenado em “a” foi **modificado para 6**.

A variável “a” permanece a mesma área de memória.

Modificações

- ❑ Podemos substituir esse valor quantas vezes desejarmos. Basta realizarmos novas atribuições. Ao longo de um programa uma variável possui seu valor modificado diversas vezes.

```
int a;  
a = 3;  
a = 6;  
a = 2;
```



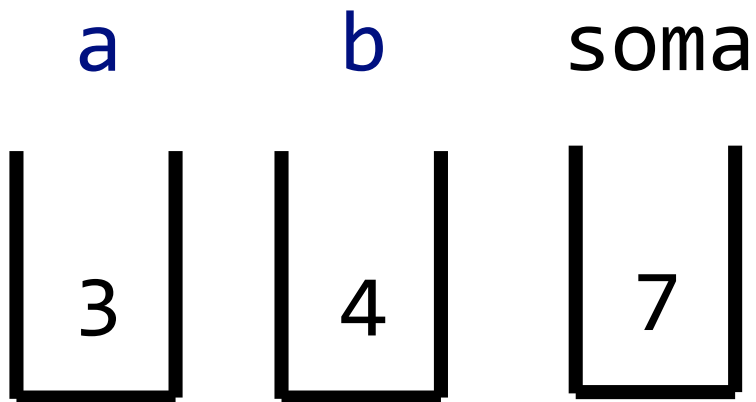
O conteúdo armazenado em “a” foi **modificado para 2**.
A variável “a” permanece a mesma área de memória.

Atribuir Resultado de Expressões

- ☐ Podemos também atribuir resultados de expressões aritméticas. Isso pode ser feito na declaração da variável ou durante a execução do programa.
- ☐ `<variável> = <expressão>`
- ☐ A variável sempre se encontra a esquerda da atribuição. A expressão fica a direita.
- ☐ A expressão pode ser grande, mas no final ela precisará resultar em um valor com o tipo adequado para ser armazenado na variável

Atribuir Resultado de Expressões

```
int a = 3;  
int b = 4;  
int soma = a + b;
```



O conteúdo armazenado em “soma” foi o resultado da soma do valor de “a” com o valor armazenado em “b”.
No fim, o que houve foi uma atribuição da mesma forma.

Nomeando Variáveis

- ❑ Existem regras que devemos seguir para nomearmos nossas variáveis.
- ❑ Essas regras, na grande maioria dos casos, são válidas em qualquer linguagem de programação.



- ✓ Sequência de **letras** e **dígitos**.
- ✓ Sempre iniciam por uma letra
 - ✓ ('**A**' – '**Z**', '**a**' – '**z**') ou
 - ✓ pelos símbolos de *underline* (**_**) ou
 - ✓ cifrão (**\$**).



- ✓ Nunca iniciam com um dígito ('**0**' – '**9**').
- ✓ Nunca contêm espaços.
- ✓ Não podem ser palavras reservadas.
- ✓ Não contém caracteres especiais

Quais desses nomes são válidos?



Quais desses nomes são válidos?

☐ A

☐ Matricula

☐ X

☐ 2X

☐ X2

☐ Nome do aluno

☐ Nome_do_aluno

☐ _teste

☐ A32B

☐ \$_classe















☐ Apartamento(201)

☐ Sala_31

☐ Sala_3.1

☐ UNIFEI

Quais desses nomes são válidos?

- | | |
|--|--|
| <input type="checkbox"/> A  | <input type="checkbox"/> _teste  |
| <input type="checkbox"/> Matricula  | <input type="checkbox"/> A32B  |
| <input type="checkbox"/> X  | <input type="checkbox"/> \$_classe  |
| <input type="checkbox"/> 2X  Iniciando com número | <input type="checkbox"/> Apartamento(201)  Parênteses |
| <input type="checkbox"/> X2  | <input type="checkbox"/> Sala_31  |
| <input type="checkbox"/> Nome do aluno  Espaço em branco | <input type="checkbox"/> Sala_3.1  Ponto “.” |
| <input type="checkbox"/> Nome_do_aluno  | <input type="checkbox"/> UNIFEI  |

Sensível a Caixa das Letras

- ❑ A linguagem C é *sensível a caixa*, do inglês *case sensitive*.
- ❑ Isso significa que ela diferencia letras maiúsculas de minúsculas.
- ❑ As seguintes variáveis são diferentes:
 - ❑ Soma ≠ SOMA ≠ SomA ≠ soma

Quais são palavras reservadas?

Palavras reservadas			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Quais são palavras reservadas?

- ❑ São palavras que o compilador “reserva” e já possuem significados pré-definidos.
- ❑ Não podemos criar variáveis ou funções que sejam palavras reservadas. Mas podemos “incluir” como subconjunto.

 `int int = 3; //Não podemos criar uma variável chamada “int”`

 `int varInt = 3 //Mas podemos usar palavras para compor outras`

Quais são palavras reservadas?

- ☐ No início precisamos “decorar” as palavras chaves/reservadas.
- ☐ Mas o compilador/ambiente ajuda nesse processo.
 - ☐ Utiliza cores diferentes em palavras reservadas.
 - ☐ Informa mensagens de erros que ajudam a detectar trechos incorretos no código.

Quais são outros tipos?

- ❑ A linguagem C oferece, além do tipo `int`, outros que nos ajudam a escrever o programa.

```
float    //Número real com precisão simples  
double   //Número real com precisão dupla  
char     // Para armazenarmos um único caractere
```

Tipos primitivos

- ❑ Estes tipos também são conhecidos como “tipos primitivos”. São fornecidos pela própria linguagem.
- ❑ Cada tipo primitivo possui uma aplicação mais adequada para ser utilizada e possui uma faixa de valores que consegue armazenar.
- ❑ Além disso, eles ocupam espaços diferentes na memória que pode variar dependendo da arquitetura do computador e da implementação do compilador.

Tipos primitivos

Tipo	Tamanho mínimo	Aplicação	Intervalo
int	2 bytes	Armazena números inteiros: (conjunto \mathbb{Z})	-32,768 a +32,767
float	4 bytes	Armazena números reais: (com parte fracionária: conjunto \mathbb{R})	Intervalo Negativo: -3.4 E+38 a -1.4 E-45 Intervalo Positivo: 1.4 E-45 a 3.4 E+38 (6 algarismos significativos)
double	8 bytes	Armazena números reais de precisão dupla: (conjunto \mathbb{R}) Possui o dobro da precisão do tipo float.	Intervalo Negativo: -1.79 E+308 a -4.94 E-324 Intervalo Positivo: 4.94 E-324 a 1.79 E+308 (10 algarismos significativos)
char	1 byte	Armazena um único caractere	-128 a +127

Tipos primitivos - Espaço

- ❑ Mesmo que a variável não esteja com o valor máximo, ela continua ocupando o mesmo espaço na memória.

```
int a = 4;  
int b = 32767;
```

- ❑ Ambas as variáveis `a` e `b` requerem a mesma quantidade de memória.
Podendo ser 2 ou 4 bytes (depende da arquitetura).
- ❑ Em outras palavras, elas ocupam o mesmo espaço.

Tipos primitivos com modificadores

- ❑ Podemos ainda utilizar modificadores para aumentar a capacidade das variáveis.
- ❑ Na linguagem C temos o modificador `long` que pode ser utilizado em variáveis do tipo `int` e `double`.
- ❑ Isso tem um preço, pois as variáveis irão requerer mais memória.

```
long int a;  
long double b;
```

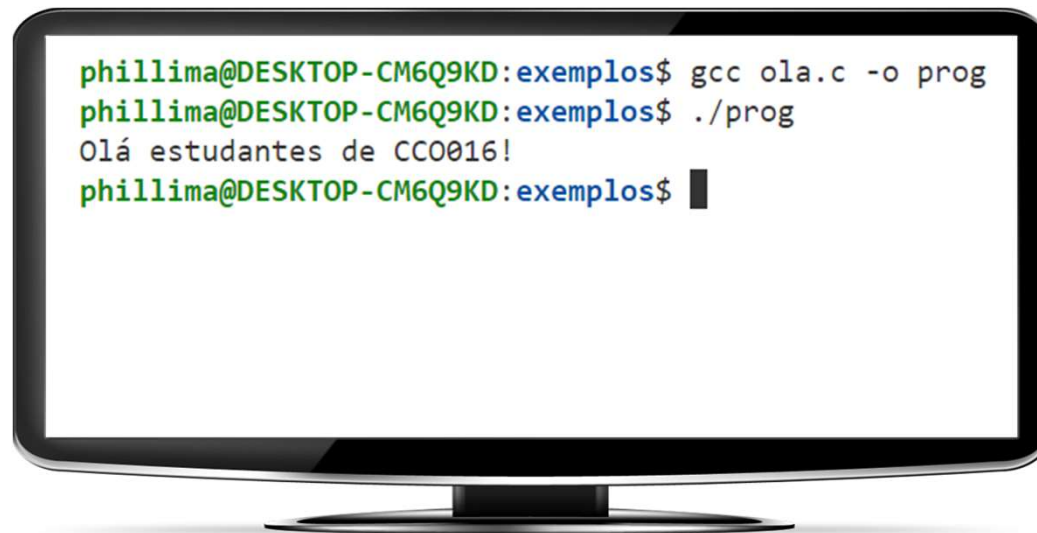
Tipos primitivos com modificadores

Tipo	Tamanho mínimo	Aplicação	Intervalo
<code>long int</code>	4 bytes	Armazena números inteiros: (conjunto \mathbb{Z})	-2147483647 a $+2147483647$
<code>long long int</code>	8 bytes	Armazena números inteiros longos: (conjunto \mathbb{Z})	-9223372036854775807 a $+9223372036854775807$
<code>long double</code>	10 bytes	Armazena números reais: (conjunto \mathbb{R})	<i>Dependente da arquitetura</i>

Imprimindo valores numéricos na tela

- ❑ Até o momento utilizamos a função “printf” apenas para mostrarmos literais.

`printf(“Olá estudantes de CC0016!\n”);`



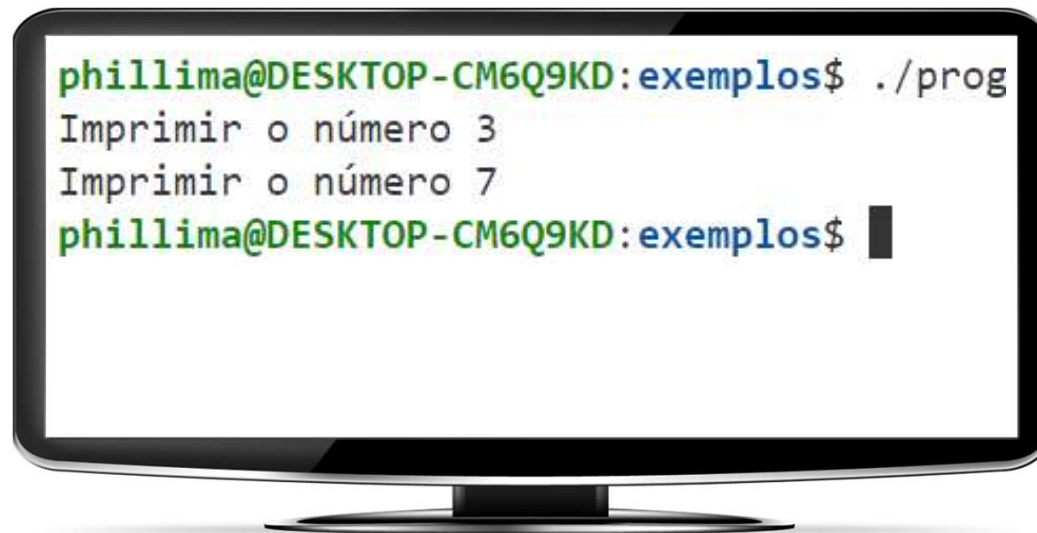
Imprimindo valores numéricos na tela

- ❑ Para imprimir valores numéricos, precisamos utilizar “marcadores” que irão definir a formatação e posição dos valores que serão impressos.
- ❑ Para formatar valores utilizamos os seguintes marcadores:
 - ❑ %d -> valor inteiro ([int](#))
 - ❑ %f -> valor real ([float](#))
 - ❑ %lf -> valor real de precisão dupla ([double](#))
- ❑ Os marcadores são inseridos durante a construção da frase, na posição que devem aparecer. Isso significa que os marcadores ficam entre aspas duplas “”.

Imprimindo valores numéricos na tela

❑ Considere a frase abaixo.

```
printf("Imprimir o número %d\n",3);  
printf("Imprimir o número %d\n",7);
```



Imprimindo valores numéricos na tela

- ❑ O marcador `%d` informa o compilador que após as aspas `""` do `printf`, teremos um valor numérico. Este valor será substituído na frase exatamente no local onde o `%d` se encontrava

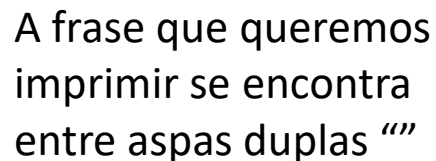
```
printf("Imprimir o número %d\n",3);
```

Imprimindo valores numéricos na tela

- ❑ O marcador `%d` informa o compilador que após as aspas `""` do `printf`, teremos um valor numérico. Este valor será substituído na frase exatamente no local onde o `%d` se encontrava

```
printf("Imprimir o número %d\n", 3);
```

A frase que queremos imprimir se encontra entre aspas duplas `""`

A blue line with an arrow at the end points from the text box to the opening double quote of the string "Imprimir o número %d\n" in the code snippet above.

Imprimindo valores numéricos na tela

- ❑ O marcador `%d` informa o compilador que após as aspas `""` do `printf`, teremos um valor numérico. Este valor será substituído na frase exatamente no local onde o `%d` se encontrava

```
printf("Imprimir o número %d\n", 3);
```

A frase que queremos imprimir se encontra entre aspas duplas `""`

O marcador deve ser colocado exatamente onde queremos imprimir o valor

Imprimindo valores numéricos na tela

- ❑ O marcador `%d` informa o compilador que após as aspas `""` do `printf`, teremos um valor numérico. Este valor será substituído na frase exatamente no local onde o `%d` se encontrava

```
printf("Imprimir o número %d\n", 3);
```

A frase que queremos imprimir se encontra entre aspas duplas `""`

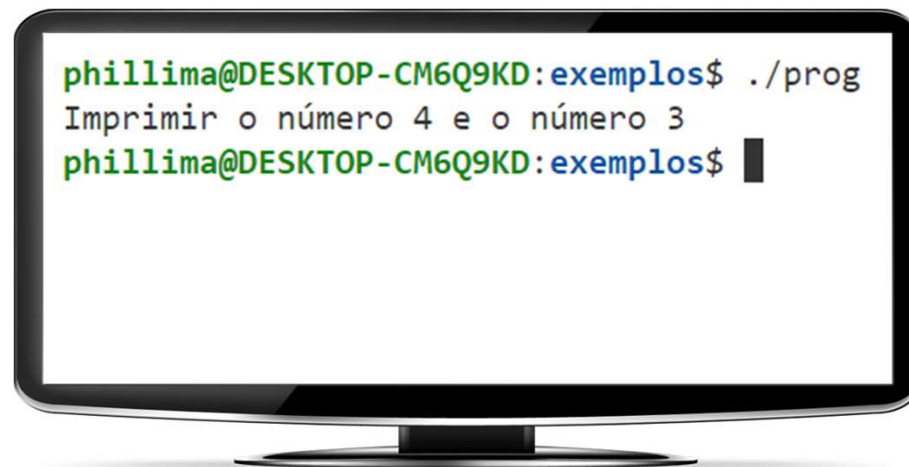
O marcador deve ser colocado exatamente onde queremos imprimir o valor

Após a vírgula (,) o compilador irá procurar um número para substituir no lugar de `%d`

Imprimindo valores numéricos na tela

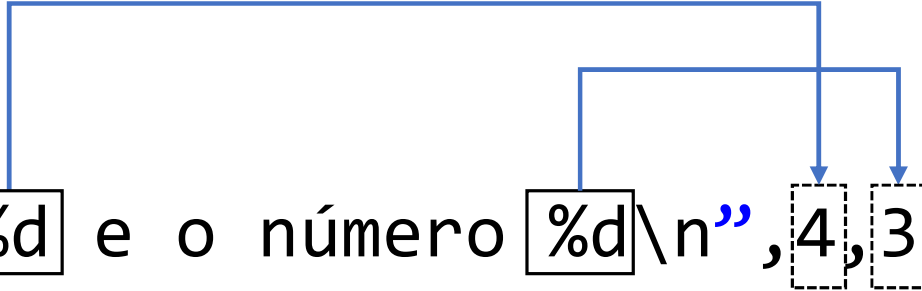
- ❑ Podemos colocar quantos marcadores desejarmos, mas precisamos, obrigatoriamente, colocar a mesma quantidade de valores para serem substituídos. Separados por vírgula (,).

```
printf("Imprimir o número %d e o número %d\n",4,3);
```



Imprimindo valores numéricos na tela

`printf("Imprimir o número %d e o número %d\n", 4, 3);`



Imprimindo valores numéricos na tela

- ❑ Nesses últimos exemplos seria mais fácil simplesmente imprimir literalmente o valor, já que é uma constante.
- ❑ O resultado será o mesmo.

`printf("Imprimir o número 4 e o número 3\n");`



Imprimindo valores numéricos na tela

- ❑ O uso real dos marcadores se encontram com as variáveis. Pois elas podem mudar o valor do meio do programa e ainda assim vamos conseguir imprimir o seu conteúdo.

`printf("Resultado da soma = %d\n", soma);`



The diagram illustrates the process of variable substitution in a printf statement. A solid blue box highlights the format specifier "%d\n", and a dashed blue box highlights the variable "soma". A blue arrow originates from the top of the solid box, extends upwards, then horizontally to the right, and finally downwards as an arrowhead pointing into the dashed box, indicating that the value of "soma" will replace "%d\n" in the output string.

- ❑ O compilador irá substituir o valor que se encontra na variável “soma” e colocar no lugar de “%d”.

Imprimindo valores numéricos na tela

- ❑ Podemos formatar os dados de outras maneiras além de números inteiros

Tipo	Especificador	Tipo	Especificador
int	%d	long int	%ld
float	%f	long long int	%lld
double	%lf	long double	%Lf

- ❑ Números reais, por padrão, apresentam 6 casas decimais. Podemos ajustar isso direto nos marcadores.
- ❑ %x.f, onde x é o número de casas que desejamos mostrar após a vírgula (,)

Programa para Somar e Mostrar o Resultado


soma.c

```
1. #include <stdio.h>
2.
3. int main() // Função Principal
4. { //Inicio
5.
6.     // Variáveis
7.     int a = 3;
8.     int b = 4;
9.     int soma = a + b;
10.
11.     printf("Resultado da soma de A e B = %d\n", soma);
12.
13.     return 0;
14. } // Fim
```

Programa para Somar e Mostrar o Resultado

soma.c

```
1. #include <stdio.h>
2.
3. int main() // Função Principal
4. { //Inicio
5.
6.     // Variáveis
7.     int a = 3;
8.     int b = 4;
9.     int soma = a + b;
10.
11.     printf("Resultado da soma de A e B = %d\n", soma);
12.
13.     return 0;
14. } // Fim
```



```
phillima@DESKTOP-CM6Q9KD:exemplos$ gcc soma.c -o prog
phillima@DESKTOP-CM6Q9KD:exemplos$ ./prog
Resultado da soma de A e B = 7
phillima@DESKTOP-CM6Q9KD:exemplos$
```

Programa para Somar e Mostrar o Resultado

soma.c

```
1. #include <stdio.h>
2.
3. int main() // Função Principal
4. { //Inicio
5.
6.     // Variáveis
7.     int a = 3;
8.     int b = 4;
9.     int soma = a + b;
10.
11.     printf("Resultado da soma de A e B = %d\n", soma);
12.
13.     return 0;
14. } // Fim
```

Área com as
variáveis.

Atribuindo o resultado
da soma de a e b

Substituindo o
valor da variável
"soma" no lugar de
%d

Operadores Aritméticos



- ❑ Além da soma, podemos também executar outras operações aritméticas

Operação	Símbolo utilizado	Exemplo
Adição	+	<code>resp = a + b;</code>
Subtração	-	<code>resp = a - b;</code>
Multiplicação	*	<code>resp = a * b;</code>
Divisão	/	<code>resp = a / b;</code>
Resto de divisão	%	<code>resp = a % b;</code>

Operadores Aritméticos



- ❑ Devemos ficar atentos a divisão. Se ambos os operandos, dividendo e divisor, forem inteiros, a saída conterá somente a parte inteira.
- ❑ Mesmo que a variável que receba o resultado seja `float/double`

```
int a = 5;  
int b = 2;  
double res = a/b;  
printf("Resultado da divisão de A por B = %f\n", res);
```

```
phillima@DESKTOP-CM6Q9KD:exemplos$ ./prog  
Resultado da divisão de A por B = 2.000000  
phillima@DESKTOP-CM6Q9KD:exemplos$
```



Esperávamos 2.5 e não 2.0

Operadores Aritméticos



- ❑ Uma alternativa é fazermos todas as operações com variáveis reais

```
double a = 5; //a e b são double
double b = 2;
double res = a/b;
printf("Resultado da divisão de A por B = %f\n", res);
```

```
phillima@DESKTOP-CM6Q9KD:exemplos$ ./prog
Resultado da divisão de A por B = 2.500000
phillima@DESKTOP-CM6Q9KD:exemplos$ █
```



Operadores Aritméticos



- ❑ Ou podemos forçar um dos operandos a serem um número real multiplicando por (1.0).

```
int a = 5;  
int b = 2;  
double res = a/(b*1.0); //Forçamos o b para real ao multiplicar por 1.0  
printf("Resultado da divisão de A por B = %f\n", res);
```

```
phillima@DESKTOP-CM6Q9KD:exemplos$ ./prog  
Resultado da divisão de A por B = 2.500000  
phillima@DESKTOP-CM6Q9KD:exemplos$ █
```

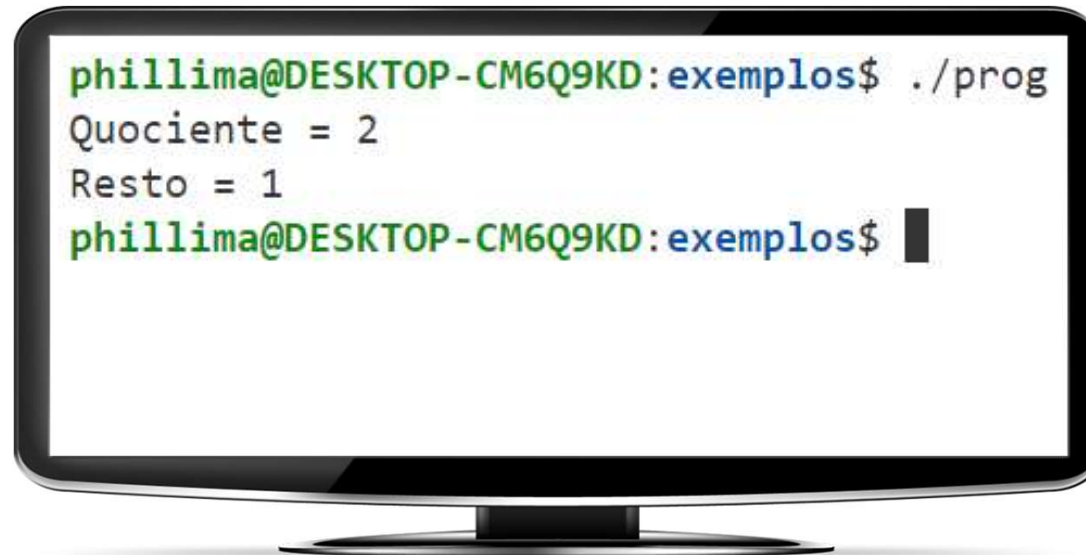


Operadores Aritméticos



- ❑ O operador `%` (resto de divisão) só é usado com números inteiros.

```
int a = 5;
int b = 2;
int quoc = a/b;
int res = a%b;
printf("Quociente = %d\nResto = %d\n", quoc,res);
```





Entrada de Datos

Entrada de dados

- ❑ Na linguagem C utilizamos a função **scanf()** para fazer a leitura de valores digitados no teclado.
- ❑ No pseudocódigo é o equivalente a função “LER”.
- ❑ A função **scanf()** requer de dois parâmetros.
 - ❑ O primeiro diz quantos valores serão lido e o tipo destes. Funciona exatamente como os marcadores que vimos para **printf()**
 - ❑ O segundo parâmetro é onde o valor digitado será salvo, isto é, qual variável irá armazenar.

Entrada de dados

- ❑ No exemplo abaixo estamos declarando a variável inteira “a”. Em seguida iremos fazer a leitura de um valor que foi digitado no teclado.
- ❑ Toda vez que o programa encontra a função **scanf()**, ele fica parado esperando um valor ser digitado e a tecla ENTER ser pressionada.
- ❑ Enquanto isso não ocorrer, o programa não segue adiante.

```
int a;  
scanf("%d", &a);
```

Entrada de dados

```
int a;  
scanf("%d", &a);
```

Indica que estamos aguardando um valor inteiro, por isso o %d. Observe que o marcador fica entre aspas ""

Indica que iremos salvar o valor lido na variável `a`.
**PERCEBA QUE PRECISAMOS COLOCAR O OPERADOR &
ANTES DO NOME DA VARIÁVEL. SEMPRE**

Entrada de dados

- ❑ Podemos ler vários valores de uma única vez, mas precisamos especificar todos os marcadores e também todas as variáveis que irão receber esses valores na ordem que aparecem.

```
int a;  
int b;  
int c;  
scanf("%d %d %d", &a,&b,&c);
```


Entrada de dados

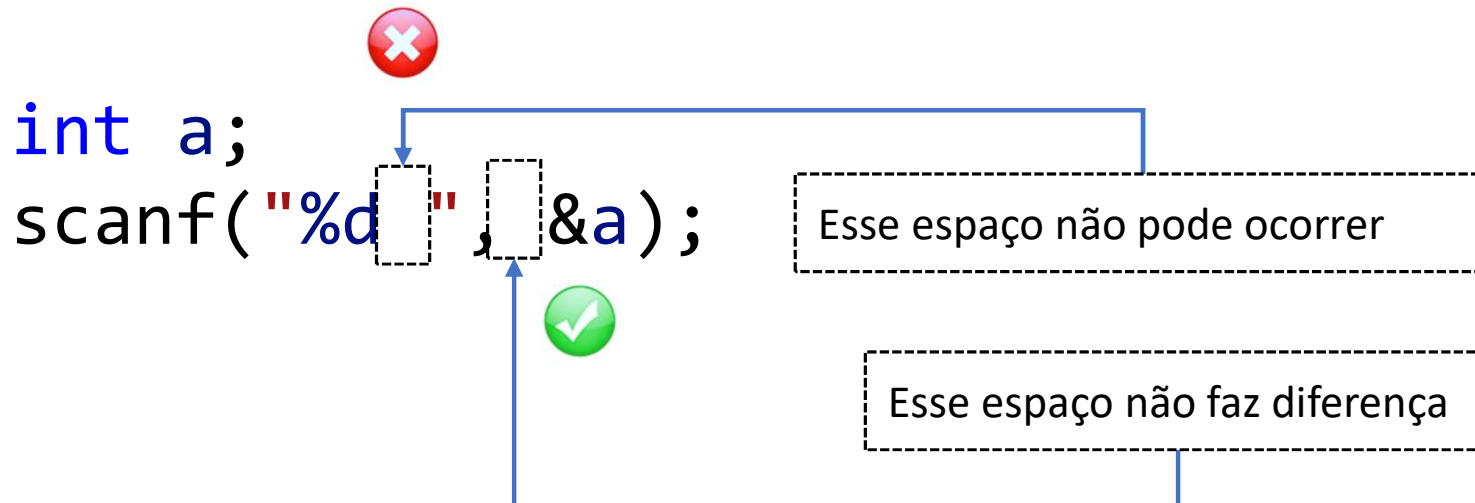
- ❑ Podemos ler vários valores de uma única vez, mas precisamos especificar todos os marcadores e também todas as variáveis que irão receber esses valores na ordem que aparecem.

```
int a;  
int b;  
int c;  
scanf("%d %d %d", &a, &b, &c);
```

The diagram illustrates the mapping between the format specifiers in the `scanf` function and the variables they read into. It shows three blue arrows originating from the first three `%d` format specifiers and pointing to the memory addresses `&a`, `&b`, and `&c` respectively. Additionally, there are three blue arrows originating from the `&a`, `&b`, and `&c` variables and pointing back to the first three `%d` format specifiers, indicating the flow of data from the input to the variables.

Entrada de dados

- ❑ Perceba que os marcadores estão dentro das aspas duplas "".
- ❑ Além disso foi colocado um espaço entre eles para melhor legibilidade
- ❑ Esse espaço, porém, não pode ocorrer se for a leitura de apenas um único valor



Entrada de dados

- ❑ Não fica agradável “tacar” a função **scanf()** sem informar o que está acontecendo. Pois o programa irá **travar** e quem estiver usando o programa não saberá o que deve fazer.
- ❑ Assim é comum combinarmos as funções printf() e scanf()
- ❑ O printf() está para MOSTRAR assim como scanf() está para LER


Entrada de dados

```
int a;  
printf("Digite um valor: ");  
scanf("%d", &a);
```

Entrada de dados

```
int a;  
printf("Digite um valor: ");  
scanf("%d", &a);
```

Informar que será
necessário digitar um
valor no teclado

A blue line with an arrowhead points from the right side of the `printf` statement in the code block to the bottom-left corner of the callout box.

Entrada de dados

```
int a;  
printf("Digite um valor: ");  
scanf("%d", &a);
```

Informar que será
necessário digitar um
valor no teclado

O programa ficará
aguardando um valor ser
digitado e pressionar ENTER

Entrada de dados

```
int a;  
printf("Digite um valor: ");  
scanf("%d", &a);
```

Informar que será
necessário digitar um
valor no teclado

O programa ficará
aguardando um valor ser
digitado e pressionar ENTER

O valor digitado será salvo na
variável `a`

Programa para ler dois valores e mostrar o resultado

soma.c

```
#include <stdio.h>

int main() // Função Principal
{ // Início

    // Variáveis
    int a, b, soma;

    printf("Digite um valor para A: ");
    scanf("%d", &a);

    printf("Digite um valor para B: ");
    scanf("%d", &b);

    soma = a + b;

    printf("Resultado da soma de A e B = %d\n", soma);

    return 0;
} // Fim
```


Programa para ler dois valores e mostrar o resultado

soma.c

```
#include <stdio.h>

int main() // Função Principal
{ // Início

    // Variáveis
    int a, b, soma;

    printf("Digite um valor para A: ");
    scanf("%d", &a);

    printf("Digite um valor para B: ");
    scanf("%d", &b);

    soma = a + b;

    printf("Resultado da soma de A e B = %d\n", soma);

    return 0;
} // Fim
```

Variáveis do mesmo tipo
podem ser declaradas na
mesma linha

Programa para ler dois valores e mostrar o resultado

soma.c

```
#include <stdio.h>
```

```
int main() // Função Principal  
{//Inicio
```

```
// Variáveis  
int a,b,soma;
```

```
printf("Digite um valor para A: ");  
scanf("%d", &a);
```

```
printf("Digite um valor para B: ");  
scanf("%d", &b);
```

```
soma = a + b;
```

```
printf("Resultado da soma de A e B = %d\n", soma);
```

```
return 0;
```

```
} // Fim
```

Variáveis do mesmo tipo
podem ser declaradas na
mesma linha

Pedimos os valores e
armazenamos nas
respectivas variáveis

Programa para ler dois valores e mostrar o resultado

soma.c

```
#include <stdio.h>
```

```
int main() // Função Principal  
{//Inicio
```

```
// Variáveis  
int a,b,soma;
```

```
printf("Digite um valor para A: ");  
scanf("%d", &a);
```

```
printf("Digite um valor para B: ");  
scanf("%d", &b);
```

```
soma = a + b;
```

```
printf("Resultado da soma de A e B = %d\n", soma);
```

```
return 0;
```

```
} // Fim
```

Variáveis do mesmo tipo
podem ser declaradas na
mesma linha

Pedimos os valores e
armazenamos nas
respectivas variáveis

Computamos a soma de a e b

Programa para ler dois valores e mostrar o resultado

soma.c

```
#include <stdio.h>
```

```
int main() // Função Principal  
{//Inicio
```

```
// Variáveis  
int a,b,soma;
```

```
printf("Digite um valor para A: ");  
scanf("%d", &a);
```

```
printf("Digite um valor para B: ");  
scanf("%d", &b);
```

```
soma = a + b;
```

```
printf("Resultado da soma de A e B = %d\n", soma);
```

```
return 0;
```

```
} // Fim
```

Variáveis do mesmo tipo
podem ser declaradas na
mesma linha

Pedimos os valores e
armazenamos nas
respectivas variáveis

Computamos a soma de a e b

Mostramos o resultado

Programa para ler dois valores e mostrar o resultado

soma.c

```
#include <stdio.h>

int main() // Função Principal
{ // Início

    // Variáveis
    int a, b, soma;

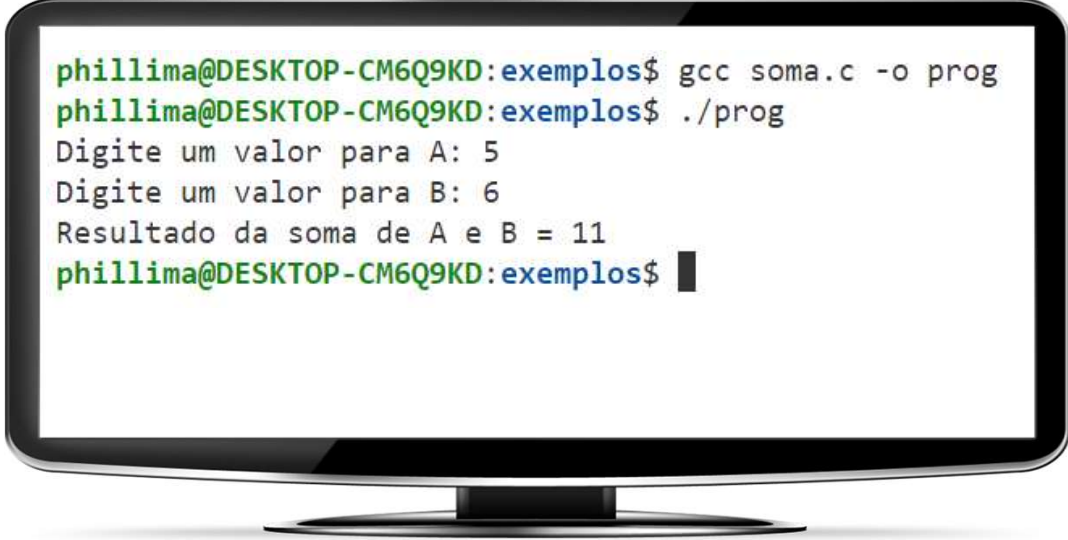
    printf("Digite um valor para A: ");
    scanf("%d", &a);

    printf("Digite um valor para B: ");
    scanf("%d", &b);

    soma = a + b;

    printf("Resultado da soma de A e B = %d\n", soma);

    return 0;
} // Fim
```



```
phillima@DESKTOP-CM6Q9KD:exemplos$ gcc soma.c -o prog
phillima@DESKTOP-CM6Q9KD:exemplos$ ./prog
Digite um valor para A: 5
Digite um valor para B: 6
Resultado da soma de A e B = 11
phillima@DESKTOP-CM6Q9KD:exemplos$
```

Programa para ler dois valores e mostrar o resultado

soma.c

```
#include <stdio.h>

int main() // Função Principal
{ // Início

    // Variáveis
    int a, b, soma;

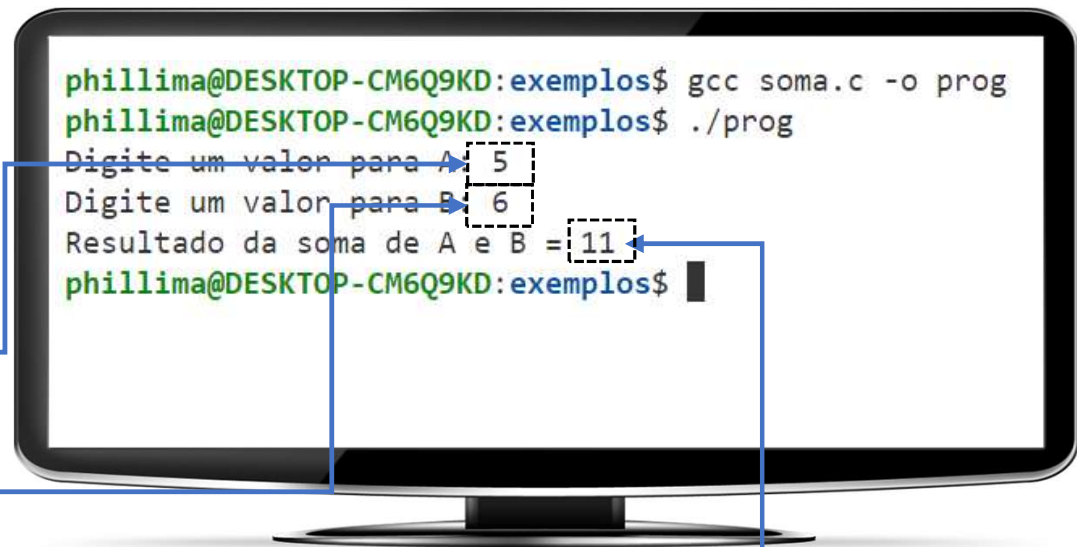
    printf("Digite um valor para A: ");
    scanf("%d", &a);

    printf("Digite um valor para B: ");
    scanf("%d", &b);

    soma = a + b;

    printf("Resultado da soma de A e B = %d\n", soma);


    return 0;
} // Fim
```





Exemplos

- ❑ **Exemplo 1:** criar um programa que leia um inteiro e um real e mostre o seu produto.
- ❑ **Exemplo 2:** criar um programa que leia um inteiro e o eleve ao cubo.
- ❑ **Exemplo 3:** criar um programa que leia uma temperatura em graus Fahrenheit e a converta para graus Celsius. A fórmula é: $^{\circ}C = (^{\circ}F - 32) * \frac{5}{9}$
- ❑ **Exemplo 4:** sabendo que uma loja está fazendo uma promoção e dando desconto de 5% em todos os itens vendidos, criar um programa que leia o preço de um produto e mostre o seu novo valor, com desconto.



Funções Matemáticas

Funções Matemáticas

- ❑ A linguagem C traz diversas funções que realizam cálculos matemáticos, como potenciação, radiciação, logaritmos, seno etc.
- ❑ O arquivo de cabeçalho onde estas funções estão definidas é chamado `math.h`.
- ❑ Assim, o código deve conter o comando `#include <math.h>` junto com `#include <stdio.h>`
- ❑ Para realizar operações aritméticas (+, -, *, /) não é necessário realizar esta inclusão.

Funções Matemáticas

- ❑ Potenciação: `pow(b, e)`
 - ❑ Eleva o argumento `b` (base) ao expoente `e`. Tanto o resultado quanto os parâmetros `b` e `e` são tratados como tendo o tipo `double`.
- ❑ Raiz quadrada: `sqrt(n)`
 - ❑ Extrai a raiz quadrada de `n`. O resultado possui o tipo `double`.
- ❑ Seno, cosseno e tangente: `sin(a)`, `cos(a)`, `tan(a)`
 - ❑ Calculam, respectivamente, o seno, cosseno e a tangente do ângulo `a`.
 - ❑ O ângulo deve ser informado em radianos.
 - ❑ O resultado possui o tipo `double`.

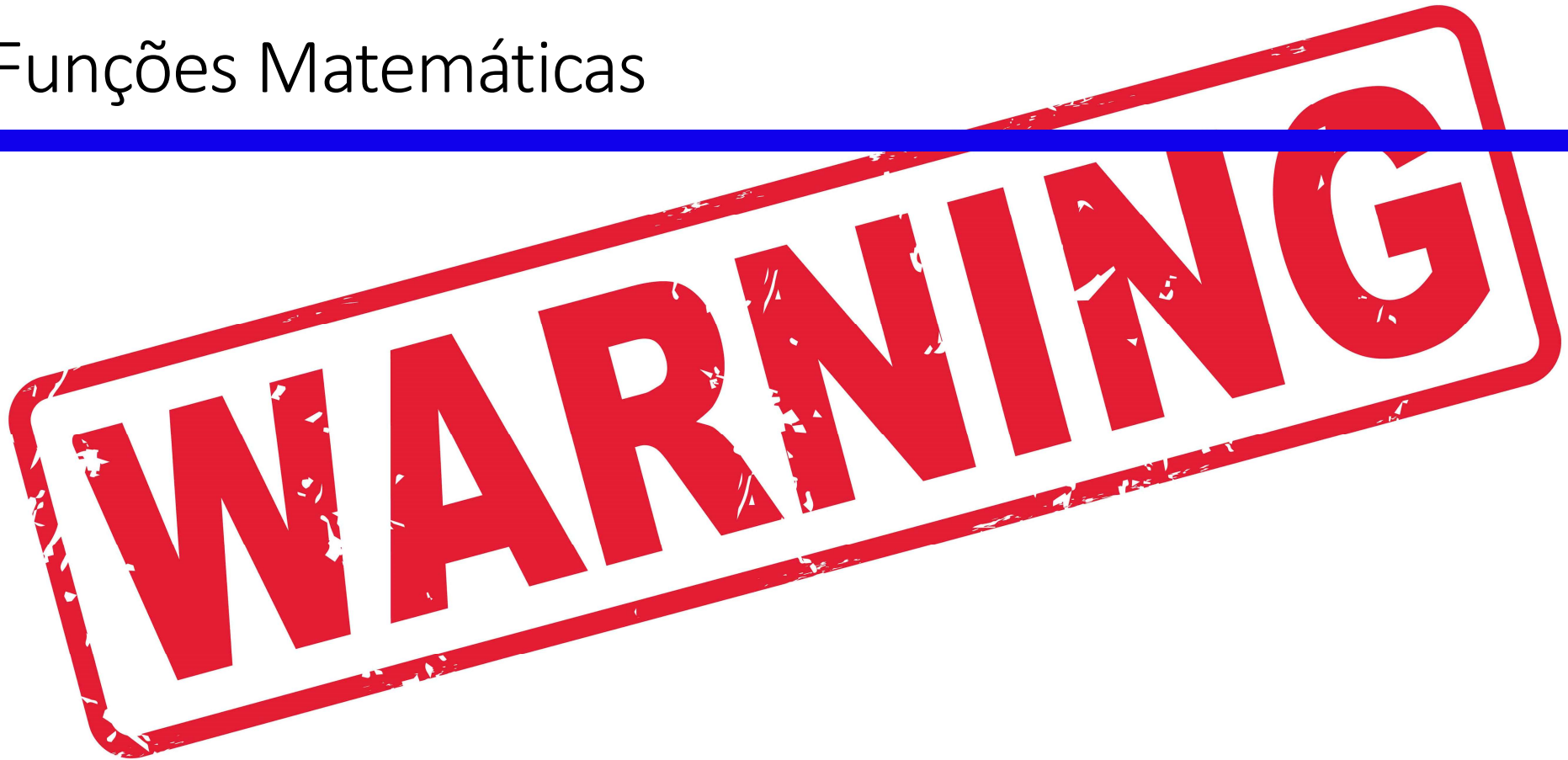
Funções Matemáticas

- ❑ As funções possuem um **valor de retorno**, que podem ser atribuídas a variáveis
- ❑ Entenda as funções matemáticas como **expressões aritméticas**.
- ❑ `<var> = <função>`
- ❑ Exemplo com potenciação:

```
double potencia = pow(2,3);
```

A função `pow(2,3)` fará o cálculo de 2^3 . O resultado será atribuído na variável `potencia`

Funções Matemáticas



- ❑ Para compilar com GCC no Linux as funções matemáticas, precisamos usar o parâmetro **-lm** no final

Programa para ler dois valores e mostrar o resultado

pot.c

```
#include <stdio.h>
#include <math.h>

int main() // Função Principal
{ // Início

    // Variáveis
    int a = 2, b = 3;
    double potencia = pow(a, b);
    printf("O número %d elevado a %d = %0.0f\n", a, b, potencia);
    return 0;
} // Fim
```

Programa para ler dois valores e mostrar o resultado

pot.c

```
#include <stdio.h>
#include <math.h>
```

Incluir a biblioteca da matemática <math.h>

```
int main() // Função Principal
{ // Início
```

A função pow() fará o cálculo da potência e irá retornar o resultado

```
    // Variáveis
    int a = 2, b = 3;
    double potencia = pow(a, b);
    printf("O número %d elevado a %d = %0.0f\n", a, b, potencia);
    return 0;
} // Fim
```

Atribuímos o resultado na variável **potencia**

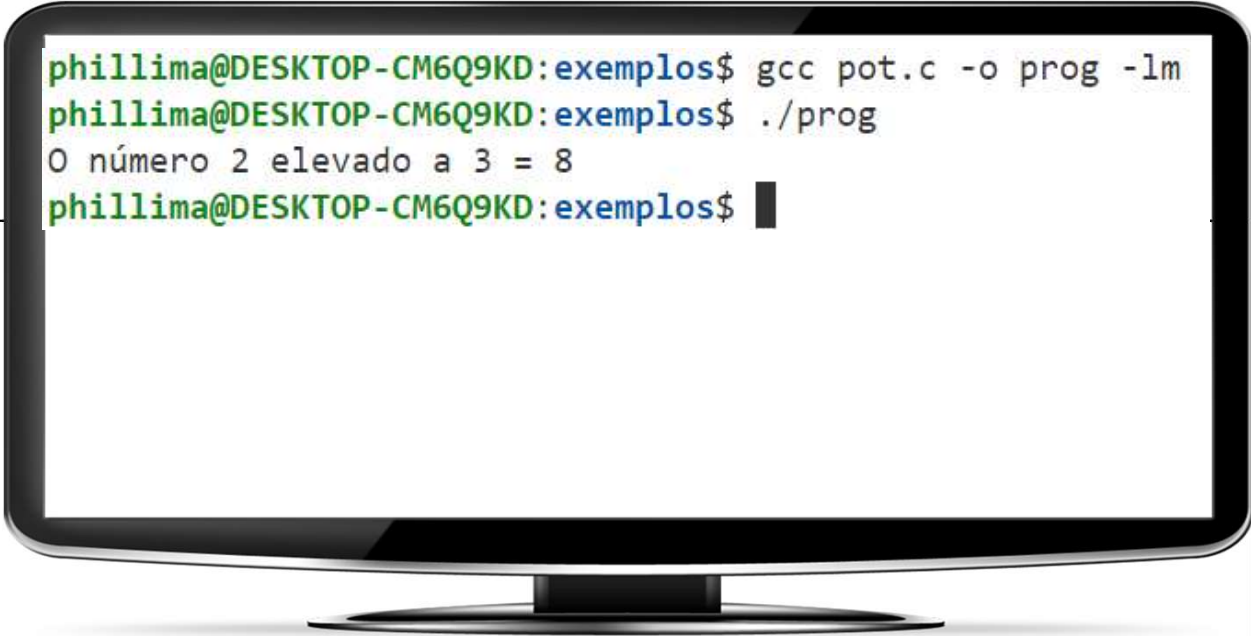
Programa para ler dois valores e mostrar o resultado

pot.c

```
#include <stdio.h>
#include <math.h>

int main() // Função Principal
{ // Início

    // Variáveis
    int a = 2, b = 3;
    double potencia = pow(a, b);
    printf("O número %d elevado a %d = %0.0f\n", a, b, potencia);
    return 0;
} // Fim
```



```
phillima@DESKTOP-CM6Q9KD:exemplos$ gcc pot.c -o prog -lm
phillima@DESKTOP-CM6Q9KD:exemplos$ ./prog
O número 2 elevado a 3 = 8
phillima@DESKTOP-CM6Q9KD:exemplos$
```


Programa para ler dois valores e mostrar o resultado

pot.c

```
#include <stdio.h>
#include <math.h>

int main() // Função Principal
{ // Início

    // Variáveis
    int a = 2, b = 3;
    double potencia = pow(a, b);
    printf("O número %d elevado a %d = %0.0f\n", a, b, potencia);
    return 0;
} // Fim
```



```
phillima@DESKTOP-CM6Q9KD:exemplos$ gcc pot.c -o prog -lm
phillima@DESKTOP-CM6Q9KD:exemplos$ ./prog
O número 2 elevado a 3 = 8
phillima@DESKTOP-CM6Q9KD:exemplos$
```

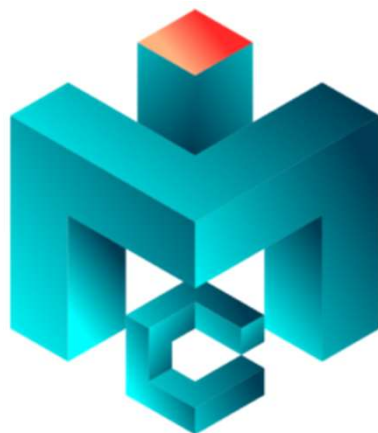
A terminal window with a black background and green text. It shows the compilation of 'pot.c' into 'prog' using 'gcc' with the '-lm' flag. The program is then executed, displaying the output 'O número 2 elevado a 3 = 8'. A blue arrow points from the '-lm' flag in the command line to a red dashed box containing the text 'Observe o -lm ao final do comando'.

Observe o -lm ao final do comando



Exemplos

- ❑ **Exemplo 5:** ler as medidas dos catetos de um triângulo retângulo e calcular o valor da hipotenusa.



Aula – 4

Variáveis e Entrada de Dados

Disciplina: CCO016 - Fundamentos de Programação

Prof: Phyllipe Lima
phyllipe@unifei.edu.br

Universidade Federal de Itajubá – UNIFEI
IMC – Instituto de Matemática e Computação